Replicated Data and Domain Decomposition Molecular Dynamics Techniques for Simulation of Anisotropic Potentials

MARK R. WILSON,[1] MICHAEL P. ALLEN,[2] MARK A. WARREN,[2] ALAIN SAURON,[3] and WILLIAM SMITH[4]

[1]*Department of Chemistry, University of Durham, South Road, Durham, DH1 3LE, UK*
[2]*H. H. Wills Physics Laboratory, Bristol, UK*
[3]*Sheffield Hallam University, Sheffield, UK*
[4]*Daresbury Laboratory, Warrington, UK*

Received 20 December 1995; accepted 26 June 1996

ABSTRACT

The implementation of parallel molecular dynamics techniques is discussed in the context of the simulation of single-site anisotropic potentials. We describe the use of both replicated data and domain decomposition approaches to molecular dynamics and present results for systems of up to 65536 Gay–Berne molecules on a range of parallel computers (Transtech i860/XP Paramid, Intel iPSC/860 Hypercube, Cray T3D). We find that excellent parallel speed-ups are possible for both techniques, with the domain decomposition method found to be the most efficient for the largest systems studied.

Introduction

In recent years, molecular dynamics (MD) has become one of the most widely used techniques for studying condensed systems. It can be applied to a wide range of problems including the simulation of liquids,[1] biological systems,[2] and the investigation of complex fluids such as liquid crystals,[3] colloids,[4] and micellar systems.[5] In most of these cases, molecular dynamics involves the solution of a set of coupled differential equations by using finite-difference techniques. In these algorithms it is the calculation of forces between individual particles which provides the main computational task. For large systems, this undertaking can use in excess of 90% of the CPU time required in the whole calculation, and can provide a major barrier to the large-scale simulations required to

*Author to whom all correspondence should be addressed.

CCC 0192-8651 / 97 / 040478-15

study complex fluids. Hence, there is a continued need to develop new algorithms and techniques to speed up the force calculation. A potential solution to the simulation of large structures is provided by parallel computers. Here the simple idea of allowing each of a series of processors to tackle part of a complex problem provides (in principle) a cost-effective solution to the difficulty posed by the force calculation. In the past, parallel computers have developed along a number of lines. These include SIMD (single instruction, multiple data) machines where a single instruction is executed simultaneously on each processor; and MIMD (multiple instruction, multiple data) machines where each processor can act separately from its neighbors. The latter is currently the most popular type of parallel machine and can be cheaply assembled from a network of individual processors each with communication links. This article concentrates on MIMD architectures and examines the simulation of anisotropic systems by molecular dynamics. The work described is based around the Gay–Berne potential [6, 7], an axially symmetric single-site potential frequently used in the simulation of liquid crystal systems. The methods described can however by easily extended to the simulation of other anisotropic potentials including biaxial systems and this will be discussed. Two parallel approaches are examined based around replicated data and domain decomposition techniques. We consider the relative performance of both methods on a number of machines and discuss the sources of inefficiency in both of the algorithms.

Simulation of Anisotropic Systems

In the current study, we are concerned with describing the motion of molecular systems which are represented by single-site anisotropic potentials and interact with each other within a periodic box. For such systems, molecular motion can be divided into the translational motion of the center of mass, and rotational motion about it. For the translational motion the net force, $\mathbf{f}_i$, on the center of mass of molecule i is given by a simple vector sum over pairwise forces between i and its neighbors:

$$\mathbf{f}_i = -\sum_j \nabla U_{ij}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) \tag{1}$$

where the anisotropic potential, U_{ij}, is a function of the vector between molecular centers, $\mathbf{r}_{ij}$, and the molecular orientations, $\mathbf{\Omega}_i, \mathbf{\Omega}_j$. The neighbor molecules, j, are usually specified with respect to a spherical cut-off which defines the range of the potential. Similarly, for rotational motion, the torque, $\boldsymbol{\tau}_i$, on molecule i is given by a vector sum of the torques, $\boldsymbol{\tau}_{ij}$, which arise from individual pairwise interactions with neighbor molecules:

$$\boldsymbol{\tau}_i = \sum_j \boldsymbol{\tau}_{ij}. \tag{2}$$

In this study, the form used for U_{ij} is taken to be the Gay–Berne potential.[6,7] This potential has been studied by a number of investigators[8,9] and provides an excellent model for the simulation of liquid crystal systems. Our choice of model, however, is not unique in the sense that the techniques developed in this study can easily be extended to other anisotropic potentials.

In the case of Gay–Berne molecules, the pair potential is axially symmetric, so eq. (2) can be rewritten as:

$$\boldsymbol{\tau}_i = \mathbf{e}_i \times \mathbf{g}_i = \mathbf{e}_i \times \sum_j \mathbf{g}_{ij} \tag{3}$$

$$\mathbf{g}_{ij} = -\nabla_{\mathbf{e}_i} U_{ij} = -\left(\frac{\partial U_{ij}}{\partial e_{ix}}, \frac{\partial U_{ij}}{\partial e_{iy}}, \frac{\partial U_{ij}}{\partial e_{iz}} \right) \tag{4}$$

where $\mathbf{e}_i$ is the vector along the long axis of molecule i. However, it is usual to avoid the evaluation of the torques in this equation and work instead with the so-called gorques, $\mathbf{g}_i$. In doing so, the equations of motion are conveniently written in the following terms[1,10]:

$$m_i \ddot{\mathbf{r}}_i = \mathbf{f}_i \tag{5}$$

$$\ddot{\mathbf{e}}_i = \frac{\mathbf{g}_i^{\perp}}{I} + \lambda_i \mathbf{e}_i \tag{6}$$

where:

$$\mathbf{g}_i^{\perp} = \mathbf{g}_i - (\mathbf{g}_i \cdot \mathbf{e}_i)\mathbf{e}_i \tag{7}$$

is the sum of the components of $\mathbf{g}_i$ perpendicular to the long axis, I is the molecular moment of inertia, and λ_i is a Lagrange multiplier. Setting $\mathbf{v} = \dot{\mathbf{r}}$ and $\mathbf{u} = \dot{\mathbf{e}}$, eqs. (5)–(6) are solved using a leap-frog algorithm from Fincham.[11] For the time-step δt:

$$\mathbf{v}_i\left(t + \frac{1}{2}\delta t\right) = \mathbf{v}_i\left(t - \frac{1}{2}\delta t\right) + \delta t \frac{\mathbf{f}_i}{m_i} \tag{8}$$

$$\delta t \lambda_i(t) = -2\mathbf{u}_i\left(t - \frac{1}{2}\delta t\right) \cdot \mathbf{e}_i(t) \quad (9)$$

$$\mathbf{u}_i\left(t + \frac{1}{2}\delta t\right) = \mathbf{u}_i\left(t - \frac{1}{2}\delta t\right) + \delta t \frac{\mathbf{g}_i^{\perp}}{I} + \delta t \lambda_i(t)\mathbf{e}_i(t) \quad (10)$$

$$\mathbf{r}_i(t + \delta t) = \mathbf{r}_i(t) + \delta t \mathbf{v}_i\left(t + \frac{1}{2}\delta t\right) \quad (11)$$

$$\mathbf{e}_i(t + \delta t) = \mathbf{e}_i(t) + \delta t \mathbf{u}_i\left(t + \frac{1}{2}\delta t\right) \quad (12)$$

$\mathbf{r}_i$, $\mathbf{v}_i$, $\mathbf{e}_i$, $\mathbf{u}_i$, $\mathbf{f}_i$, and $\mathbf{g}_i^{\perp}$ must all be stored during the simulation. In the case of the parallel algorithms described in the next section, the dominant computational task involves the calculation of $\mathbf{f}_i$ and $\mathbf{g}_i$ via eqs. (1) and (4); and the main communication costs are related to the passing of these variables from one processor to another at appropriate points in the force calculation. The secondary task of integrating the equation of motion via eqs. (8)–(12) is relatively cheap for serial code, but becomes significant when the force calculation is parallelized.

Parallel Techniques

REPLICATED DATA METHOD

The replicated data approach to molecular dynamics is outlined in the schematic diagram in Figure 1. This method is a typical example of the data parallel approach which can be implemented for a large number of problems. Each processor in a network stores its own copy of the simulation data, and the same simulation program runs simultaneously on each node. For inexpensive operations there is little to be gained from parallelization, so the computational effort is simply duplicated on each node. Time savings accrue where expensive program operations can be divided between the individual processors and carried out in parallel. However, some of the time saved by these operations is then negated by the need to circulate any changed data around the network of nodes. The key to successful parallelization then rests with maximizing time spent in the parallel sections of the code and at the same time minimizing the amount of data interchanged in the time-consuming communication steps.

The key numerical task involves the calculation of forces and gorques using eqs. (1) and (4). In principle, all pair interactions, $\mathbf{f}_{ij}$, $\mathbf{g}_{ij}$, could be considered in computing $\mathbf{f}_i$ and $\mathbf{g}_i$. However, in practice, this is usually not required for two reasons. First, $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$ (Newton's third law) and the calculations of $\mathbf{g}_{ij}$ and $\mathbf{g}_{ji}$ involve many common subexpressions. Second, the pair interactions are normally only included within a minimum cutoff distance. A convenient path forward is provided by introducing the Brode–Ahlrichs form of the Verlet neighbor list.[12] While this algorithm was originally devised to avoid the production of short vectors for upper triangular matrices, it also provides a useful way of introducing approximate load-balancing for parallel machines. The algorithm works as follows. For each particle i, the vectors, $\mathbf{r}_{ij}$, are computed for the following values of j:

for $j = i + 1$ to $i + N_{\text{offset}}$
with $j \leftarrow j - N$ for $j > N$

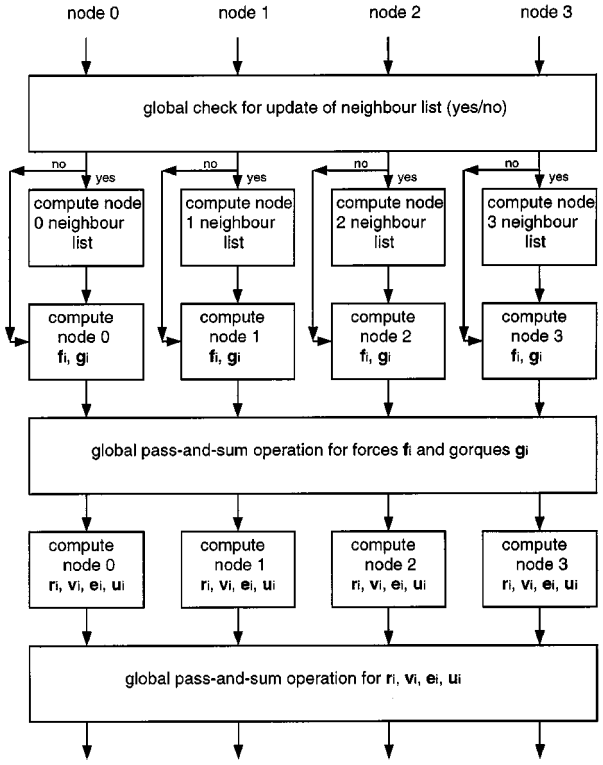

FIGURE 1. Flow diagram for a single time-step of the replicated data algorithm (for four nodes). The Verlet neighbor list generation, the force calculation, and the integration algorithm are all carried out in parallel with each node taking successive values of i. Global pass-and-sum operations are required at the end of the force calculation and at the end of the integration step. A global check for the need to update the neighbor list is carried out at the start of each time-step.

where:

$$N_{\text{offset}} = \begin{cases} \text{int}(N/2) & \text{for odd } i \\ N/2 & \text{for even } i,\ i \le N/2 \\ N/2 - 1 & \text{for even } i,\ i > N/2 \end{cases}$$

[where the operation int$(N/2)$ takes the integer part of $N/2$]. This process is done in parallel with each node being allocated a value of i in turn. A Verlet neighbor list[13] is then compiled in the usual way on each node for distances $r_{ij} < r_L$, where the Verlet cutoff, r_L, is slightly larger than the pair-potential cutoff, r_c. The Verlet lists can then be used to compute the forces and gorques in parallel, using the following procedure:

1. $\mathbf{f}_i$ and $\mathbf{g}_i$ are set to zero for all molecules on each node.
2. Each node calculates the values of $\mathbf{f}_{ij}$ and $\mathbf{g}_{ij}$ for the ij pairs in its own neighbor list subject to the condition $r_{ij} < r_c$, and computes $\mathbf{f}_i$ and $\mathbf{g}_i$ from eqs. (1) and (4).
3. A global *pass-and-sum* operation is carried out, whereby the incomplete values of $\mathbf{f}_i$ and $\mathbf{g}_i$ on each node are summed and distributed to each node.
4. Finally, the values of $\mathbf{g}_i^{\perp}$ are calculated on each node from eq. (7).

Once each node has a copy of the correct values of $\mathbf{f}_i$ and $\mathbf{g}_i^{\perp}$, the equations of motion can be integrated. The parallel algorithm steps through eqs. (8)–(12) with each node taking successive values of i as before. Following this a second global *pass-and-sum* operation is required to sum and distribute the values of $\mathbf{r}_i$, $\mathbf{v}_i$, $\mathbf{e}_i$, $\mathbf{u}_i$, to each node before a further force calculation can take place. For any network of processors it is important to carry out the global sum operation with the minimum cost in terms of communication time. The efficiency of this operation depends strongly on the number of nodes in a network and on the nature of the interprocessor connectivity. However, the most efficient global *pass-and-sum* operations usually occur for hypercube geometries. For a hypercube of size 2^d processors, only d (the minimum possible) overlapped messages are required for the whole operation. This compares to $2^d - 1$ overlapped messages for a ring of 2^d processors where each processor is connected to two neighbors. In this work, hypercube geometries have been used exclusively. The following pseudo-code summarizes the *global pass-and-sum* operation and was implemented on all machines to operate on the M elements of the array **ARRAY**. It uses a work array **WORK** of equal size to **ARRAY** and executes simultaneously for nodes with identifiers **IDNODE** between 0 and $2^d - 1$.

```
for b = 0 to d − 1
  get the neighbor node id NEIGH by flipping
      the b^th bit of IDNODE
  blocked_send: to send the array ARRAY to
      neighbor node NEIGH
  blocked_receive: to receive the array WORK
      for i = 1 to M
    ARRAY(i) ← ARRAY(i) + WORK(i)
  endfor
endfor
```

Hence, the replicated data method is quick and easy to implement, and only minor changes to the usual scalar procedures are required.

DOMAIN DECOMPOSITION METHOD

The domain decomposition strategy is a well-known approach to the simulation of large numbers of particles interacting via short range forces. The basic data structure is outlined in Figure 2 for a two-dimensional system. The system as a whole is divided into a series of regions of equal size, with each region handled by a separate compute node. The regions themselves are divided into cells of equal size with the number of cells maximized subject to the condition that none of the cell sides are smaller than the cutoff on U_{ij}. The values of $\mathbf{r}_i$, $\mathbf{v}_i$, $\mathbf{e}_i$, $\mathbf{u}_i$, $\mathbf{f}_i$, $\mathbf{g}_i^{\perp}$ for a particular particle are all stored locally on the processor which handles that particle's region of space. Molecules in a particular cell will interact only with particles in the same or adjacent cells. This means that, for a majority of particles in a region, no reference needs to be made to particles in other regions which are being handled by a different node on the network. Only particles which reside in cells on the boundary of a particular region need special treatment. For these particles interactions must be considered between other particles on the edge of neighboring regions and this procedure involves message-passing between neighboring nodes. However, the amount of data required to be passed is much smaller than that needed for a global sum. This potentially means that this algorithm provides substantial time savings over the replicated data

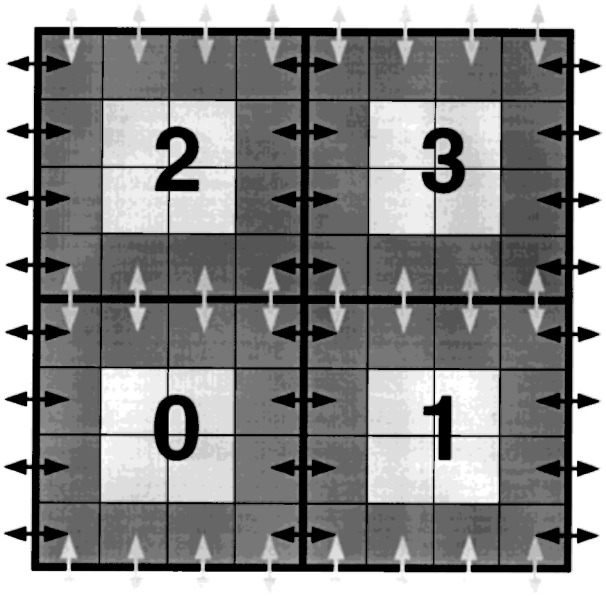

FIGURE 2. Diagram showing the data structure in the domain decomposition algorithm for a two-dimensional system on a four-node computer. The simulation box is divided into a set of regions. Molecule data for each region is handled by separate processors. Regions are divided into cells, with sides greater than r_c. Cells on the boundary of a region (dark shading) must pass $\mathbf{r}_i$, $\mathbf{e}_i$ data to neighboring nodes during the force calculation (first in the *x*-direction then in the *y*-direction), and pass $\mathbf{r}_i$, $\mathbf{e}_i$, $\mathbf{v}_i$, $\mathbf{u}_i$ data for molecules which move between regions at the end of the integration step.

approach. The basic steps in the domain decomposition algorithm are described below.

1. Two maps are established. The first map represents the connectivity between the individual cells within a region. The second map establishes the connectivity between different regions themselves. Both operations take into account the cubic or cuboidal periodic boundary conditions which are imposed on the system.
2. For each region a list is compiled of molecules which reside in the boundary cells of the region. The positions and orientations of these molecules must be exported to neighboring nodes for the force calculation to take place. This is done with six separate communication steps:
   - Send $\mathbf{r}_i, \mathbf{e}_i$ to $+x$ neighbor node and receive data from $-x$ neighbor.
   - Send $\mathbf{r}_i, \mathbf{e}_i$ to $-x$ neighbor node and receive data from $+x$ neighbor.
   - Send $\mathbf{r}_i, \mathbf{e}_i$ to $+y$ neighbor node and receive data from $-y$ neighbor.
   - Send $\mathbf{r}_i, \mathbf{e}_i$ to $-y$ neighbor node and receive data from $+y$ neighbor.
   - Send $\mathbf{r}_i, \mathbf{e}_i$ to $+z$ neighbor node and receive data from $-z$ neighbor.
   - Send $\mathbf{r}_i, \mathbf{e}_i$ to $-z$ neighbor node and receive data from $+z$ neighbor.

   The x, y, z communication steps cannot be undertaken simultaneously as it is necessary to carry out extra sorting steps between data passes to ensure that molecules from cells at the corners or along the edges of a region are properly exported to neighboring regions. For example, in the case of the corner of a region, the molecule data must undergo x, y, and z data transfers before it has reached all its destinations.
3. The $N_r + N_e$ molecules in a region (resident molecules plus molecule orientations and positions imported from other nodes) are then used to generate a linked list[14] for use in the force calculation.
4. The force calculation then proceeds using the linked list to evaluate forces only for those molecules N_r which properly reside on each node.
5. Each node calculates new values for $\mathbf{r}_i$, $\mathbf{v}_i$, $\mathbf{e}_i$, $\mathbf{u}_i$, using eqs. (8)–(12) for its resident atoms only.
6. Molecules whose positions move out of the region of space allocated to their node must be exported to neighboring nodes. Again, this is done with a sequence of x, y, z communication steps as in step 2. (On this occasion, values of $\mathbf{v}_i$, $\mathbf{u}_i$ must be passed and received in addition to values of $\mathbf{r}_i$, $\mathbf{e}_i$.) The molecule data which has been passed to other nodes is then deleted from the stored list (on each node) and the received molecule data is added to this list.
7. Steps 2–6 can then be iterated.

On each step it is usual to calculate the total potential and kinetic energies. This is done via a *global pass-and-sum* operation as in the replicated data code. However, the cost of this is negligible because only small amounts of data need to be transported.

The domain decomposition strategy is much harder to implement than the replicated data approach, requiring a major change in strategy from

scalar molecular dynamics. However, the reduced communication costs described above provide for potentially large speed-ups of molecular dynamics on massively parallel machines. This has led to the successful implementation of variants of this approach in a number of studies of Lennard–Jones systems.[15–17]

EXTENSION TO BIAXIAL SYSTEMS

For biaxial systems, a convenient replacement for eqs. (8)–(12) is provided by a quaternion form of the leap-frog algorithm.[1,18] Here the vectors $\mathbf{e}_i$, $\mathbf{u}_i$ are replaced by quaternions $\mathbf{Q}_i = (q_{i0}, q_{i1}, q_{i2}, q_{i3})$ specifying the molecular orientation, and angular momenta vectors $\mathbf{L}_i$. This algorithm fits neatly into the replicated data and domain decomposition schemes discussed above, with $\mathbf{Q}_i$ directly replacing $\mathbf{e}_i$, and $\mathbf{L}_i$ replacing $\mathbf{u}_i$ in the message passing steps. The biaxial algorithm, however, requires one extra variable to specify molecular orientation, and this must be passed during interprocessor communications. Consequently, it is slightly less efficient than the linear molecule algorithms described above.

Results

The results for our implementation of the replicated data strategy are presented in Table I. We consider two separate parallel machines: a Transtech Paramid system based on 50-MHz i860 XP processors and T805 D-30 Transputer links, and a Cray T3D system with 150-MHz DEC Alpha 21064 processors with high bandwidth communication links. For both machines, the manufacturer's implementation of PVM[19] was used to provide a common structure to the simulation code. We present benchmark data for cubic systems of Gay–Berne molecules of sizes $N = 256$, 2048, 16384, and an $N = 65536$ system in a cuboidal box with sides in the ratios 1 : 2 : 2. Calculations use the parameterization of the Gay–Berne potential described in ref. 9 at a reduced density of $\rho^* = 0.3$ and a reduced temperature of $T^* = 1.5$. This corresponds to the isotropic phase. A cutoff of $r_c/\sigma_0 = 4.0$ was used for the Gay–Berne potential along with a Verlet list cutoff of $r_L/\sigma_0 = 4.3$. We have excluded I/O from the benchmarks along with any initialization costs associated with the algorithm. Both these factors are negligible for long simulation runs. In the case of both machines the benchmarks illustrate that considerable speed-ups are available from parallelization, but that parallel speed-up depends critically on system size. For the Transtech machine, the force calculation parallelizes well with performance gains available for up to 8 processors with 256 molecules, and 16 processors for 2048 molecules. The fall-off in parallel performance here is caused by the large communication costs associated with the global pass-and-sum required for forces and gorques. As the number of processors rises this part of the algorithm starts to become communication bound. However, the parallelization of the Verlet list suffers no such problems. A separate list is compiled for each node and thus there are no communication costs associated with this part of the algorithm. Deviations from 100% efficiency arise simply due to incomplete load balancing between individual processors, and this situation improves dramatically with increasing system size. Finally, in examining the integration algorithm itself, we note that parallelization leads to a net *reduction* in speed. The execution of eqs. (8)–(12) is so quick that any speed-ups due to parallelization are completely outweighed by the global pass-and-sum operations required for the quantities $\mathbf{r}_i, \mathbf{v}_i, \mathbf{e}_i, \mathbf{u}_i$.

It is important to look at the overall time spent in each section of the algorithm. For 256 molecules on one node, the direct force calculation represents 96% of the CPU usage. For such small systems, the Verlet list actually provides only small savings in CPU time, as the majority of possible pair interactions are included in the list. However, as system size increases, the percentage of time spent in the neighbor list calculation increases. The relative amounts of CPU time spent in these two routines is influenced by the difference in the two cutoff distances $r_L - r_c$. The results of this work suggest that the difference in parallel efficiency of these two parts of the algorithm will lead to an optimum value of r_L which will change with processor number, and will differ from one parallel machine to another. Specifically for a particular state-point, the optimum value of $r_L - r_c$ depends on the number of particles, N, the number of nodes chosen, and the ratio of communication time to processor speed, $R = T_c/T_p$, on the specific parallel machine used.

The performance difference between Alpha and i860 processors is evident from a comparison of the figures in Table I. However, the most striking difference between the Transtech and the T3D is provided by the communications. In the force routine on the T3D, parallel speed-ups are available

TABLE I.
Benchmark Figures for the Replicated Data Strategy on a Cray T3D and a Transtech Paramid (using i860 / XP Processors and T805 Transputer Communication Links).[a]

		Cray T3D			Paramid		
N	Nodes	Forces calculation	Verlet list	Integration algorithm	Forces calculation	Verlet list	Integration algorithm
256	1	9.4	0.106	0.17	24.9	0.85	0.19
256	2	4.8	0.054	0.19	13.8	0.49	1.4
256	4	2.5	0.027	0.24	9.0	0.29	2.8
256	8	1.5	0.014	0.29	8.0	0.23	4.7
256	16	0.88	0.008	0.34	8.2	0.25	6.2
256	32	0.65	0.005	0.40			
256	64	0.55	0.004	0.45			
256	128	0.53	0.004	0.51			
256	256	0.56	0.004	0.58			
2048	1	93.1	13.9	1.8	259.1	56.4	1.8
2048	2	47.3	6.9	1.7	140.7	29.2	10.9
2048	4	24.6	3.5	2.1	85.8	15.2	21.0
2048	8	13.5	1.7	2.1	68.3	7.8	35.0
2048	16	8.7	0.87	3.1	67.3	4.1	47.0
2048	32	6.2	0.44	3.6			
2048	64	4.2	0.22	3.0			
2048	128	3.8	0.11	3.3			
2048	256	3.8	0.06	3.8			
16384	1	2169	1293	17.1			
16384	2	1017	648.8	14.5			
16384	4	512.6	327.5	17.3			
16384	8	266.6	163.8	18.8			
16384	16	151.4	81.8	24.9			
16384	32	91.4	41.0	28.9			
16384	64	63.3	20.6	32.8			
16384	128	40.2	10.4	27.2			
16384	256	35.5	5.3	29.5			
65536	16	1784	1327	83.0			
65536	32	937.1	663.9	91.9			
65536	64	521.5	333.5	106.0			
65536	128	313.1	167.8	113.9			
65536	256	216.5	85.0	125.8			

[a]Times are given in seconds for 50 MD steps.

for up to 128 nodes with $N = 256$ and $N = 2048$ molecules and for 256 processors with $N = 16384$ molecules (and above). On the largest system of $N = 65536$ molecules, the performance of the Verlet list code is also very good, suffering only slightly from imperfect load-balancing even for large numbers of processors. We notice however that, even on the T3D, the integration code is communication bound. For $N = 65536$ molecules, scalar integration can be carried out in 73.79 seconds for 50 steps; which is significantly faster than parallel integration. However, for a 256-processor calculation, this means that 20% of the total CPU time is spent in carrying out scalar integration. This inevitably limits the parallel efficiency of the whole algorithm as the number of nodes increases. In Figure 3 we plot the best overall speed-ups achieved for the replicated data approach using scalar integration. The total calculation time $T_s = T_p + T_c = T_p(1 + R)$ in Figure 3 depends critically on the relative performance of communications and processor speed for the parallel machine used. Smith[15] has shown that, for large systems, R will increase as $1/N$ ($R \propto 2^{d+1}d/N$). We therefore expect a regime where, for large N, communication costs become negligible and the algorithm achieves

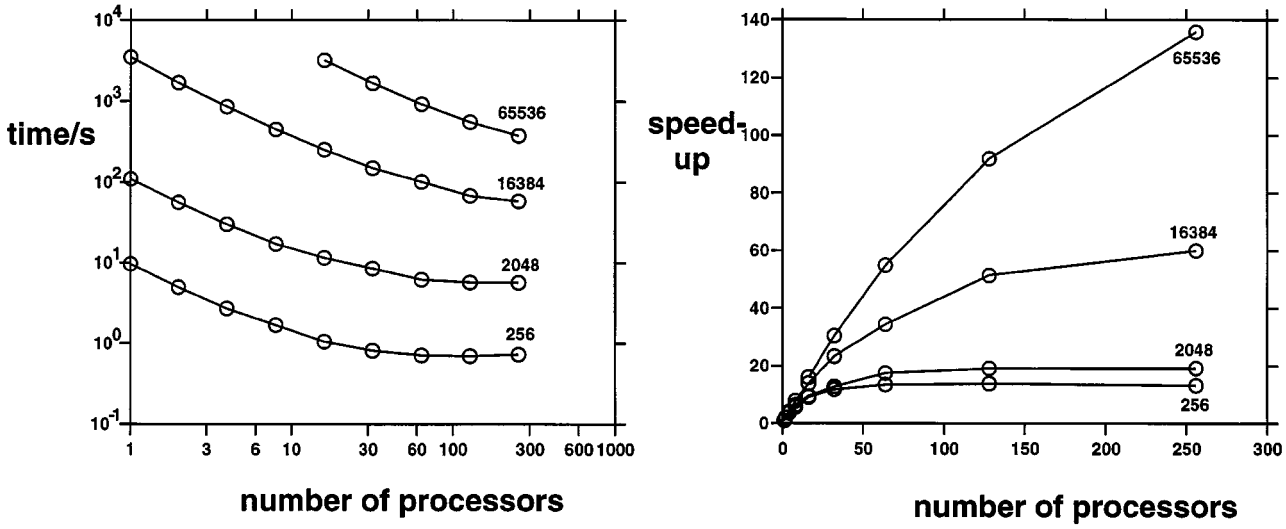

FIGURE 3. Results for the replicated data algorithm on a Cray T3D. Curves are labeled by system size N (number of molecules). (a) Total simulation times for 50 steps (log scale) versus number of processors (log scale). (b) Speed-up relative to one processor versus number of processors.

linear scaling. This is not achieved in Figure 3, although we clearly see improving parallel efficiency with increasing system size. For 65536 molecules on 256 processors a speed-up of approximately 136 was achieved. For even larger systems we would expect even better performances. However, for very large systems the replicated data algorithm quickly runs into memory limits. To a first approximation the dimension of the Verlet list per node is $4\pi r_L^3 \rho^* N/(6 \times 2^d)$, and this provides a severe restriction on the size of system which can be studied.

In Table II we list the benchmark results for the domain decomposition algorithm. Results are presented for an Intel iPSC/860 Hypercube in addition to the Transtech and Cray results. The systems used were identical to those discussed above. As above, parallel speed-ups for each machine are largely determined by T_c/T_p. Consequently, the T3D has the best performance, followed by the Intel Hypercube, and lastly the Transtech Paramid. However, the reduced amounts of interprocessor communications in the domain decomposition algorithm, mean that excellent parallel results were achieved on all three machines. We note that (as before) the force calculation dominates the CPU usage, taking between 92% (in the worst case for 256 molecules on an eight-node Transtech Paramid) and 98% of time for an MD cycle. In contrast to the replicated data program the integration part of the algorithm works extremely efficiently. Here the equations of motion are integrated separately on each node, with the only major source of inefficiency provided by the need to pass data between adjacent nodes when molecules move from one region to another. The latter represents a negligible cost, and poses no serious limits to simulation on a large numbers of processors.

The force calculation itself scales extremely well with increasing numbers of processors. Here the major loss in efficiency is provided by the need to export data between adjacent nodes for molecules which lie in cells at a face, edge, or corner of a region. This process causes some duplication in the force calculation as well as contributing communication costs to the overall timings. However, in this case, the communication costs are considerably less than in the global add-and-sum routine which is required for the replicated data work. We note that, in none of the cases examined, does the algorithm become communications bound. For the work presented here the limit to the number of processors occurs simply when the number of cells per region falls to one. For large systems, the boundary cells in a region represent only a small part of the costs of the force evaluation. So, in the limit of large N, the ratio $R = T_c/T_p$ tends to zero and the domain decomposition algorithm should scale linearly with the number of nodes.

In Figure 4 we plot the speed-ups achieved for this algorithm on the Cray T3D. The results are extremely good. In all cases we reach a nearly linear speed-up with the number of nodes. For 65536 molecules on 256 nodes a speed-up of 202 $\times$

TABLE II.
Benchmark Figures for the Domain Decomposition Strategy on a Cray T3D and on i860-Based Parallel Computers.[a]

	Nodes	Cells	Cray T3D		Transtech Paramid		Intel Hypercube	
N	(and node grid)	in regions	Force calculation	MD cycle	Force calculation	MD cycle	Force calculation	MD cycle
256	1 (1 : 1 : 1)	2 : 2 : 2	27.6	28.3				
256	8 (2 : 2 : 2)	1 : 1 : 1	2.9	3.1	7.3	7.9	23.3	24.4
2048	1 (1 : 1 : 1)	4 : 4 : 4	198.3	203.3				
2048	8 (2 : 2 : 2)	2 : 2 : 2	26.3	27.0	35.6	36.6	132.8	135.9
2048	16 (2 : 2 : 4)	2 : 2 : 1	11.8	12.3	20.2	21.3	75.0	77.1
2048	32 (2 : 4 : 4)	2 : 1 : 1	5.7	6.0				
2048	64 (4 : 4 : 4)	1 : 1 : 1	3.1	3.4				
16384	1 (1 : 1 : 1)	8 : 8 : 8	1462	1505				
16384	8 (2 : 2 : 2)	4 : 4 : 4	197.3	202.2				
16384	16 (2 : 2 : 4)	4 : 4 : 2	102.8	105.5				
16384	32 (2 : 4 : 4)	4 : 2 : 2	53.6	55.0				
16384	64 (4 : 4 : 4)	2 : 2 : 2	28.0	28.8				
16384	128 (4 : 4 : 8)	2 : 2 : 1	14.4	14.9				
16384	256 (4 : 8 : 8)	2 : 1 : 1	6.5	7.0				
65536	1 (1 : 1 : 1)	8 : 16 : 16	5659	5831				
65536	8 (2 : 2 : 2)	4 : 8 : 8	787.2	808.7				
65536	16 (2 : 2 : 4)	4 : 8 : 4	424.4	435.6				
65536	32 (2 : 4 : 4)	4 : 4 : 4	197.3	202.3				
65536	64 (4 : 4 : 4)	2 : 4 : 4	104.9	107.5				
65536	128 (4 : 4 : 8)	2 : 4 : 2	54.0	55.5				
65536	256 (4 : 8 : 8)	2 : 2 : 2	28.0	29.0				

[a]Mean times are given in seconds for 50 MD steps.

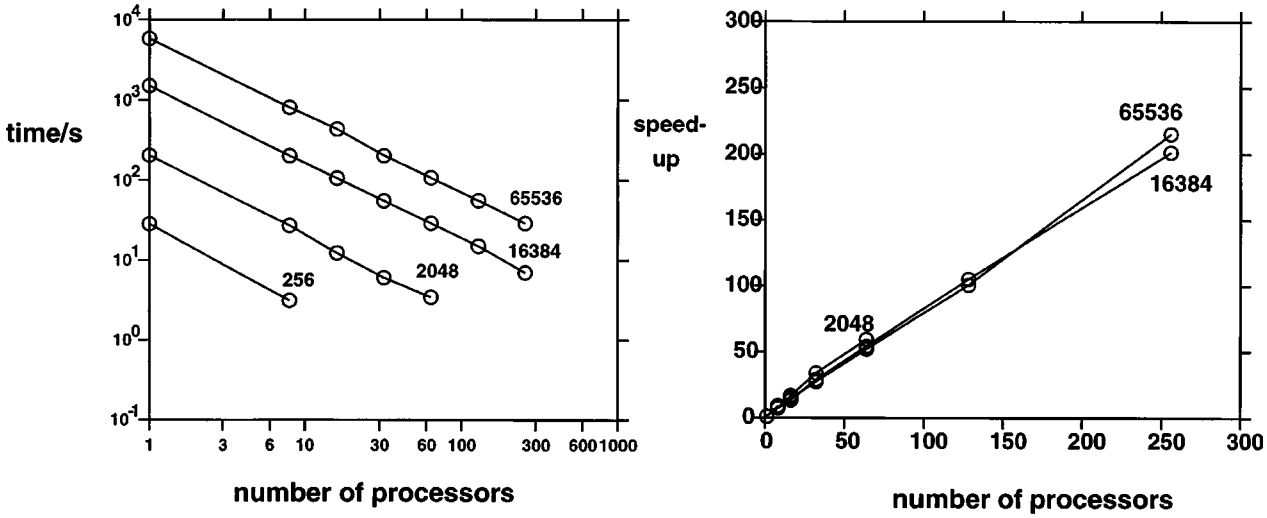

FIGURE 4. Results for the domain decomposition algorithm on a Cray T3D. Curves are labeled by system size *N* (number of molecules). (a) Total simulation times for 50 steps (log scale) versus number of processors (log scale). (b) Speed-up relative to one processor versus number of processors.

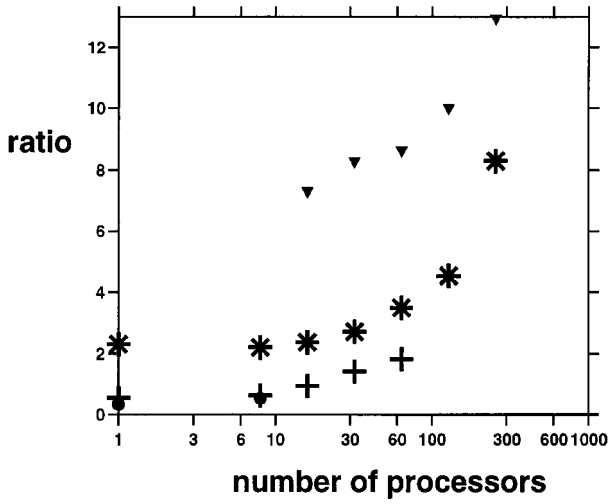

FIGURE 5. Ratio of simulation speeds domain decomposition algorithm: replicated data algorithm. Circles: $N = 256$ molecules; crosses: $N = 2048$ molecules; stars: $N = 16384$ molecules; triangles: $N = 65536$ molecules.

relative to the single processor time was achieved. This is particularly impressive because only two cells were used in each region, meaning that every particle is exported to another region during the force calculation. We would expect near 100% speed-ups in the limit where the number of cells per region became large. Again the size of the system which can be studied is limited by available memory. However, our implementation of the domain decomposition algorithm requires considerably less memory than the replicated data method. Finally, it is interesting to compare the overall efficiencies of the two approaches. This is done is Figure 5 where we consider the ratio of simulation times for identical systems using the two simulation approaches. In the case of small N, Verlet lists work more efficiently than linked lists. This is reflected in Figure 5 where for the two smallest systems (256 and 2048 molecules) the replicated data approach is the most efficient for up to eight nodes. For all other cases the domain decomposition strategy wins out. For the largest system on 256 nodes, the domain decomposition strategy provides a performance enhancement of a factor of 13 over the replicated data code.

Conclusions

We have examined the implementation of replicated data and domain decomposition algorithms for the Gay–Berne mesogen on a number of parallel machines. We find excellent speed-ups for the domain decomposition approach on a Cray T3D reaching a speed-up factor of 202 for 256 processors and 65536 molecules. We also find that this algorithm works extremely well even for relatively small numbers of molecules. In none of the cases examined does the algorithm become communications bound. This is an important result because in simulating complex fluids it is not just the simulation size which is of primary importance: the length of simulation is often critical too. Here it is simply the dimensions of the system (relative to the potential cutoff) that physically limit the number of processors which can be fitted onto a grid of cells.

The replicated data algorithm presented here works well for small numbers of processors, but, as expected, scales less efficiently than the domain decomposition program. In the limit of large systems the replicated data method also becomes efficient. However, on a particular machine, the available memory may easily be exceeded before the algorithm starts to approach maximum efficiency. In addition, at large N, the parallel Verlet list approach used here proves to be considerably less efficient than the parallel link list used in the domain decomposition method. The simplicity of the replicated data approach is, however, attractive. The algorithm presented in this work involves only minor changes from a typical scalar code, and is much easier to program than the domain decomposition algorithm. It is also relatively easy to calculate globally averaged quantities (such as the radial distribution function) via replicated data strategies. These are difficult to carry out via domain decomposition methods, particularly when the required distances exceed the width of the region handled by each compute node. Finally, it is worth stressing that the domain decomposition approach can only be used for potentials with relatively short-range cutoffs, and this provides a major limitation to this method. However, charged systems are often important in complex fluid studies, and for such simulations the usual approach involves a Ewald sum evaluation and a calculation of all the pair forces in the system. This is incompatible with the standard domain decomposition approach, but works well within the replicated data method.[20] Furthermore, the extra cost associated with Ewald sum calculations decreases the ratio of communication time to computational time, increasing the efficiency of this algorithm beyond the results presented in this

study. We therefore expect that both algorithms presented here will prove useful in the future study of anisotropic systems.

Acknowledgments

The authors thank the UK High Performance Computing Initiative for the award of computer time on the Cray T3D in Edinburgh. M.A.W. thanks the UK Engineering and Physical Sciences Research Council (EPSRC) for the award of a CASE research studentship (1992–1995). M.R.W. thanks Lancaster University for providing time on the Transtech Paramid system at Lancaster; and the EPSRC for contributing to the cost of this machine. Thanks are due to the EPSRC Collaborative Computational Project No. 5 for providing time on an iPSC/860 Intel Hypercube at Daresbury Laboratory.

References

1. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press, Oxford, 1987, Ch. 3.
2. J. A. McCammon and S. C. Harvey, *Dynamics of Proteins and Nucleic Acids*, Cambridge University Press, Cambridge, 1987.
3. M. P. Allen and M. R. Wilson, *J. Comput.-Aided Molec. Design*, **3**, 335 (1989).
4. P. Linse, *J. Chem. Phys.*, **94**, 8227 (1991).
5. K. Esselink, P. A. J. Hilbers, S. Karaborni, J. I. Siepmann, B. Smit, *Molec. Simul.*, **14**, 259 (1995).
6. B. J. Berne and P. Pechukas, *J. Chem. Phys.*, **56**, 4213 (1972).
7. J. G. Gay and B. J. Berne, *J. Chem. Phys.* **74**, 3316 (1981).
8. G. R. Luckhurst, R. A. Stephens, and R. W. Phippen, *Liquid Cryst.*, **8**, 451 (1990).
9. E. de Miguel, L. F. Rull, M. K. Chalam, and K. E. Gubbins, *Molec. Phys.*, **74**, 405 (1991).
10. K. Singer, A. Taylor, and J. V. L. Singer, *Molec. Phys.*, **33**, 1757 (1977).
11. D. Fincham, *CCP5 Quarterly*, **12**, 47 (1984).
12. S. Brode and R. Ahlrichs, *Comput. Phys. Commun.*, **42**, 51 (1986).
13. L. Verlet, *Phys. Rev.*, **159**, 98 (1967).
14. R. W. Hockney and J. W. Eastwood, *Computer Simulations Using Particles*, McGraw-Hill, New York, 1981.
15. W. Smith, *Computer Phys. Commun.*, **62**, 229 (1991).
16. D. C. Rapaport, *Comput. Phys. Rep.*, **9**, 1 (1988).
17. M. R. S. Pinches, D. J. Tildesley, and W. Smith, *Molec. Simul.*, **6**, 51 (1991).
18. D. Fincham, *CCP5 Quarterly*, **2**, 6 (1981).
19. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM 3 User Guide*, ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, TN, 1993.
20. W. Smith, *Comput. Phys. Commun.*, **67**, 392 (1992).